

REMARKS

Applicant respectfully requests reconsideration and allowance of the subject application. Claims 10, 18, and 26 are amended. New claims 41 and 42 are added. Claims 1-42 are pending in this application.

35 U.S.C. § 112

Claims 10 and 26 stand rejected under 35 U.S.C. §112, first paragraph.

In the March 5 Office Action, it was asserted that it is not clear whether “ (x_i-1) ” in claims 10 and 26 should be read with both the “i” and the “-1” as subscripts or not. As part of this response, claims 10 and 26 have been amended to clarify that both the “i” and the “-1” should be read as subscripts. Additionally, the specification has been amended to clarify that both the “i” and the “-1” should be read as subscripts. Support for these amendments can be found, for example, at page 18, lines 10-15.

In the March 5 Office Action, it was also asserted that it is not clear what the “+=” operator in claims 10 and 26 stands for, and that for examination purposes it is assumed to stand for: $Cks = Cks + f(x_i) \text{ XOR } x_i$. Applicant respectfully submits that the “+=” operator is well-known to those skilled in the art, and that the assumption in the March 5 Office Action is accurate. As an example of the “+=” operator, accompanying this response is a copy of pages from Lippman, Stanley B., “C++ primer”, 2nd edition, AT&T Bell Laboratories, 1991, pp. 71-72, which discusses the “+=” operator.

Thus, for at least these reasons, Applicant respectfully submits that amended claims 10 and 26 comply with 35 U.S.C. §112, second paragraph.

Applicant respectfully requests that the §112 rejections be withdrawn.

35 U.S.C. § 103

Claims 1-4, 6-7, 10-20, 22-23, and 26-40 stand rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,256,777 to Ackerman (hereinafter “Ackerman”) in view of U.S. Patent No. 5,054,787 to Richardson (hereinafter “Richardson”). Applicant respectfully submits that claims 1-4, 6-7, 10-20, 22-23, and 26-40 are not obvious over Ackerman in view of Richardson.

Ackerman is directed to a method and apparatus for debugging of optimized machine code, using hidden breakpoints (see, Title). As discussed in the Abstract of Ackerman, Ackerman describes a debugging method wherein a debug information file is constructed which includes information that identifies changes of variable value assignments to registers at plural steps of program. The information further includes data that identifies any change of sequence of machine code instructions from the sequence of source code instructions that gave rise to the machine code instructions. Using such information, hidden breakpoints are inserted into the machine code (wherein a hidden breakpoint enables access to an instruction to either store a variable value from an identified register or to move to a machine code instruction that corresponds in order to a source code instruction that gave rise to the machine code instruction). Thereafter, the program is executed under control of a debug program and, upon encountering a hidden breakpoint, automatically either stores the variable value that exists in the identified register or moves to execute a machine code instruction that is indicated

by the hidden breakpoint. The actions carried out in response to encountering the hidden breakpoint are invisible to the user.

Richardson is directed to a portable validation unit for an electronic gaming system, particularly a BINGO system (see, Abstract). A 33-byte EBC record, as well as a check byte, is transferred from an electronic handset to a validation unit (see, col. 6, lines 3-6, and col. 17, lines 37-39 and 56-57). The EBC record includes various data, such as a card number, pattern number, game number, and level number (see Fig. 2D). The check byte is matched with a checksum in a register which has been accumulating a summation of all the 33 input bytes (see, col. 17, lines 60-63). The result should be zero, if the check byte is to match the checksum (see, col. 17, lines 63-65). If the result is not zero, indicating an error, the communications routine is repeated (see, col. 17, lines 65-68).

In contrast, amended claim 1 recites, in part:

identifying a plurality of key instructions in a function;
inserting into the function, for each of the plurality of key instructions, an extra instruction that modifies a register based at least in part on the corresponding key instruction;
identifying a set of inputs to the function; and
determining a checksum for the function based at least in part on mapping contents of the register to the set of inputs.

Applicant respectfully submits that there is no disclosure or suggestion in Ackerman or Richardson of determining a checksum for the function based at least in part on mapping contents of the register to the set of inputs as recited in amended claim 1.

In the March 5 Office Action at p. 4, it is asserted that Richardson teaches identifying a set of input to a function and determining a checksum for the

function based at least in part on mapping contents of the register to the set of inputs. Applicant respectfully disagrees with this assertion.

Richardson, as discussed above, discloses that the check byte is matched with a checksum in a register which has been accumulating a summation of all the 33 input bytes (see, col. 17, lines 60-63). There is nothing in Richardson that discloses or suggests determining a checksum for a function based at least in part on mapping contents of the register to the set of inputs. In claim 1, the register is modified by extra instructions inserted into the function. There is no such modification of a register in Richardson. Rather, the checksum in Richardson is simply the summation of all the 33 input bytes. There is no mention or discussion in Richardson of any extra instructions modifying the register that is accumulating the summation of all the 33 input bytes. As such, Applicant respectfully submits that Richardson cannot disclose or suggest identifying a set of inputs to the function and determining a checksum for the function based at least in part on mapping contents of the register to the set of inputs as recited in claim 1.

With respect to Ackerman, it is noted in the March 5 Office Action at p. 4 that Ackerman does not teach identifying a set of inputs to the function and determining a checksum for the function based at least in part on mapping contents of the register to the set of inputs. Thus, Applicant respectfully submits that Ackerman is not cited as curing, and does not cure, the deficiencies of Richardson.

Thus, given that neither Ackerman nor Richardson discloses or suggests identifying a set of inputs to the function and determining a checksum for the function based at least in part on mapping contents of the register to the set of

inputs as recited in claim 1, Applicant respectfully submits that the combination of Ackerman and Richardson cannot disclose or suggest identifying a set of inputs to the function and determining a checksum for the function based at least in part on mapping contents of the register to the set of inputs as recited in claim 1. For at least these reasons, Applicant respectfully submits that claim 1 is allowable over Ackerman in view of Richardson.

Given that claims 2-4, 6-7, and 10-11 depend from claim 1, Applicant respectfully submits that claims 2-4, 6-7, and 10-11 are likewise allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 1.

With respect to claim 12, Applicant respectfully submits that Ackerman in view of Richardson does not disclose or suggest generating a checksum on bytes of a digital good without reading the bytes as recited in claim 12. In the March 5 Office Action at p. 7 it was asserted that:

In regards to claim 12, the combination of Ackerman and Richardson teaches generating a checksum on bytes of a digital good without reading the bytes (see Richardson col. 17, lines 60-67).

Applicant respectfully disagrees with this assertion.

It appears that Richardson is being relied on in the March 5 Office Action as teaching generating a checksum on bytes of a digital good without reading the bytes as recited in claim 12. Richardson, as discussed above, discloses that the check byte is matched with a checksum in a register which has been accumulating a summation of all the 33 input bytes (see, col. 17, lines 60-63). Thus, Applicant respectfully submits that Richardson is reading the bytes by accumulating a summation of the input bytes in the register. The checksum in Richardson is

simply the summation of all the 33 input bytes. Thus, as Richardson is reading the bytes over which the checksum is calculated, Applicant respectfully submits that Richardson does not disclose or suggest generating a checksum on bytes of a digital good without reading the bytes as recited in claim 12.

Applicant respectfully submits that Ackerman is not cited as curing, and does not cure, the deficiencies of Richardson as discussed above.

Thus, for at least these reasons, Applicant respectfully submits that claim 12 is allowable over Ackerman in view of Richardson.

Given that claims 13-15 depend from claim 12, Applicant respectfully submits that claims 13-15 are likewise allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 12.

With respect to claim 16, Applicant respectfully submits that, similar to the discussion above regarding claim 1, Ackerman in view of Richardson does not disclose or suggest the inserting, identifying, and determining of claim 16. For at least these reasons, Applicant respectfully submits that claim 16 is allowable over Ackerman in view of Richardson.

Given that claims 17-20, 22-23, and 26-28 depend from claim 16, Applicant respectfully submits that claims 17-20, 22-23, and 26-28 are likewise allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 16.

With respect to claim 29, claim 29 recites, in part:

a production server equipped with an oblivious checking protection tool that is used to augment the original program for protection purposes, the production server being configured to identify a plurality of segments in the original program and apply oblivious checking to each of the plurality of segments.

Applicant respectfully submits that Ackerman in view of Richardson does not disclose or suggest a production server as recited in claim 29.

In the March 5 Office Action at p. 7, Richardson is cited as teaching the application of oblivious checking. Applicant respectfully disagrees with this assertion and submits that Richardson does not disclose or suggest a production server configured to apply oblivious checking to each of a plurality of segments in an original program as recited in claim 29.

Richardson, as discussed above, discusses generating a checksum for a record being transferred from an electronic handset to a validation unit. This is a checksum on the 33 bytes of the EBC record, not a checksum on segments of a program. Applicant respectfully submits that there is no mention of applying oblivious checking to each of a plurality of segments of a program in Richardson. Thus, Applicant respectfully submits that Richardson cannot disclose or suggest a production server configured to apply oblivious checking to each of a plurality of segments in an original program as recited in claim 29.

With respect to Ackerman, it is noted in the March 5 Office Action at p. 7 that Ackerman does not teach an oblivious checking protection tool used for the application of oblivious checking to each of the plurality of segments. Thus, Applicant respectfully submits that Ackerman is not cited as curing, and does not cure, the deficiencies of Richardson.

Thus, given that neither Ackerman nor Richardson discloses or suggests a production server configured to apply oblivious checking to each of a plurality of segments in an original program as recited in claim 29, Applicant respectfully submits that the combination of Ackerman and Richardson cannot disclose or

suggest a production server configured to apply oblivious checking to each of a plurality of segments in an original program as recited in claim 29. For at least these reasons, Applicant respectfully submits that claim 29 is allowable over Ackerman in view of Richardson.

Given that claims 30-33 depend from claim 29, Applicant respectfully submits that claims 30-33 are likewise allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 29.

With respect to claim 34, claim 34 recites, in part:

a client to store and execute the protected program, the client being configured to evaluate the protected program to determine whether the protected program has been tampered with.

Applicant respectfully submits that Ackerman in view of Richardson does not disclose or suggest a client as recited in claim 34.

As discussed above, Ackerman is directed to a method and apparatus for debugging of optimized machine code, using hidden breakpoints. Ackerman describes a more efficient method for debugging optimized machine code (see, col. 2, lines 33-36), not determining whether a protected program has been tampered with. There is no mention in the debugging of Ackerman of determining whether a protected program has been tampered with. Thus, Applicant respectfully submits that Ackerman cannot disclose or suggest a client configured to evaluate a protected program to determine whether the protected program has been tampered with as recited in claim 34.

Richardson, as discussed above, describes using a checksum on the 33 bytes of the EBC record being transferred from an electronic handset to a validation unit. Richardson describes checking whether the 33 bytes of the EBC

record is transferred without error, not determining whether a protected program has been tampered with. There is no mention in Richardson of determining whether a protected program has been tampered with. Thus, Applicant respectfully submits that Richardson cannot disclose or suggest a client configured to evaluate a protected program to determine whether the protected program has been tampered with as recited in claim 34.

Thus, given that neither Ackerman nor Richardson discloses or suggests a client configured to evaluate a protected program to determine whether the protected program has been tampered with as recited in claim 34, Applicant respectfully submits that the combination of Ackerman and Richardson cannot disclose or suggest a client configured to evaluate a protected program to determine whether the protected program has been tampered with as recited in claim 34. For at least these reasons, Applicant respectfully submits that claim 34 is allowable over Ackerman in view of Richardson.

Given that claims 35-37 depend from claim 34, Applicant respectfully submits that claims 35-37 are likewise allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 34.

With respect to claim 38, Applicant respectfully submits that, similar to the discussion above regarding claim 34, Ackerman in view of Richardson does not disclose or suggest generating a checksum value for a segment of a digital good based at least in part on both a set of inputs to the segment and the content of a register that results from applying the set of inputs to the segment; comparing the generated checksum value to a stored checksum value corresponding to the segment; and determining that the digital good has been tampered with if the

generated checksum value does not match the stored checksum value as recited in claim 38.

Given that claims 39-40 depend from claim 38, Applicant respectfully submits that claims 39-40 are likewise allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 38.

Claims 5 and 21 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Ackerman in view of Richardson in further view of U.S. Patent No. 5,809,306 to Suzuki et al. (hereinafter "Suzuki"). Applicant respectfully submits that claims 5 and 21 are not obvious over Ackerman and Richardson in further view of Suzuki.

With respect to claim 5, claim 5 depends from claim 1 and Applicant respectfully submits that claim 5 is allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 1. Furthermore, Applicant respectfully submits that Suzuki is not cited as curing, and does not cure, the deficiencies of Ackerman in view of Richardson discussed above with respect to claim 1. Thus, for at least these reasons, Applicant respectfully submits that claim 5 is allowable over Ackerman in view of Richardson and Suzuki.

With respect to claim 21, claim 21 depends from claim 16 and Applicant respectfully submits that claim 21 is allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 16. Furthermore, Applicant respectfully submits that Suzuki is not cited as curing, and does not cure, the deficiencies of Ackerman in view of Richardson discussed above with respect to claim 16. Thus, for at least these reasons, Applicant

respectfully submits that claim 21 is allowable over Ackerman in view of Richardson and Suzuki.

Claims 8-9 and 24-25 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Ackerman in view of Richardson in further view of U.S. Patent No. 6,085,029 to Kolawa et al. (hereinafter "Kolawa"). Applicant respectfully submits that claims 8-9 and 24-25 are not obvious over Ackerman and Richardson in further view of Kolawa.

With respect to claims 8 and 9, claims 8 and 9 depend from claim 1 and Applicant respectfully submits that claims 8 and 9 are allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 1. Furthermore, Applicant respectfully submits that Kolawa is not cited as curing, and does not cure, the deficiencies of Ackerman in view of Richardson discussed above with respect to claim 1. Thus, for at least these reasons, Applicant respectfully submits that claims 8 and 9 are allowable over Ackerman in view of Richardson and Kolawa.

With respect to claims 24 and 25, claims 24 and 25 depend from claim 16 and Applicant respectfully submits that claims 24 and 25 are allowable over Ackerman in view of Richardson for at least the reasons discussed above with respect to claim 16. Furthermore, Applicant respectfully submits that Kolawa is not cited as curing, and does not cure, the deficiencies of Ackerman in view of Richardson discussed above with respect to claim 16. Thus, for at least these reasons, Applicant respectfully submits that claims 24 and 25 are allowable over Ackerman in view of Richardson and Kolawa.

Applicant respectfully requests that the §103 rejections be withdrawn.

New Claims

New claims 41 and 42 are added.

With respect to new claim 41, new claim 41 depends from claim 1 and Applicant respectfully submits that new claim 41 is allowable over the cited references at least because of its dependency on claim 1. Furthermore, Applicant respectfully submits that the cited references do not disclose or suggest one or more computer readable media as recited in claim 1, wherein the determining comprises determining the checksum so that if the function is changed the checksum will also change as recited in new claim 41. For at least these reasons, Applicant respectfully submits that new claim 41 is allowable over the cited references.

With respect to new claim 42, new claim 42 depends from claim 16 and Applicant respectfully submits that new claim 42 is allowable over the cited references at least because of its dependency on claim 16. Furthermore, Applicant respectfully submits that the cited references do not disclose or suggest a method as recited in claim 16, wherein the determining comprises determining the checksum value so that if the segment is changed the checksum value will also change as recited in new claim 42. For at least these reasons, Applicant respectfully submits that new claim 42 is allowable over the cited references.


Conclusion

Claims 1-42 are in condition for allowance. Applicant respectfully requests reconsideration and issuance of the subject application. Should any matter in this

case remain unresolved, the undersigned attorney respectfully requests a telephone conference with the Examiner to resolve any such outstanding matter.

Respectfully Submitted,

Date: 6/4/04

By: 
Allan T. Sponseller
Reg. No. 38,318
(509) 324-9256

Stanley B. Lippman

C++ Primer

2nd Edition

Library of Congress Cataloging-in-Publication Data

Lippman, Stanley B.

C++ Primer / by Stanley B. Lippman -- 2nd ed.

p. cm.

Includes index.

ISBN 0-201-54848-8

1. C++ (Computer program language) I. Title. II. Title: C plus
plus primer

QA76.73.C15L57 1991

005.13'3--dc20

91-18434

CIP



Copyright © 1991 by AT&T Bell Laboratories.

Reprinted with corrections June, 1993

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

This book was typeset in Palatino and Courier by the author, using a Linotron 100 phototypesetter and a DEC VAX 8550 computer running UNIX® System V, Release 2.

UNIX is a registered trademark of AT&T. MS-DOS is a registered trademark of Microsoft Corporation. DEC and VAX are trademarks of Digital Equipment Corporation. Ada is a trademark of the Ada Joint Program Office, Department of Defense, United States Government.

11 12 13 14 15 MA 969594

BEST AVAILABLE COPY

use
or
or < expr
or <= expr
or > expr
or >= expr
pr == expr
pr != expr
xpr && expr
xpr expr

Logical Operators

n that would make the evalu-

&&
=]))

an object. Applying the mem-
s always trouble. The first logi-
qually troublesome is an out-of-
guards against that possibility.
then the first two operands return

to true if its operand has a value
example,

at end

0.
sion of style. For example,

(!found)

is clear in its meaning: *not found*. Is it as clear, however, what the following condition tests?

```
!strcmp( string1, string2 )
```

`strcmp()` is a C library routine that compares its two string arguments for equality. A return value of 0 indicates that the two strings are equal. Use of the logical NOT notation in this case may actually obscure the condition under test. This is better written as

```
if ( strcmp( string1, string2 ) == 0 ) // ...
```

2.4 Assignment Operators

The left operand of the assignment operator ("=") must be an lvalue. The effect of an assignment is to store a new value in the left operand's associated storage. For example, given the following three definitions:

```
int i;  
int *ip;  
int ia[ 4 ];
```

the following are all legal assignments:

```
ip = &i;  
i = ia[ 0 ] + 1;  
ia[ *ip ] = 1024;  
*ip = i * 2 + ia[ i ];
```

The result of an assignment operator is the value of the expression that is assigned to its left operand. The data type of its result is the type of its left operand.

Assignment operators can be concatenated provided that each of the operands being assigned is of the same general data type. For example,

```
main()  
{  
    int i, j;  
    i = j = 0; // ok: each assigned 0  
    // ...  
}
```

and `j` are each assigned 0. The order of evaluation is right to left.

Assignment operator concatenation also allows for notational compression, as in the following conditional test of the value assigned to the character variable `ch`:


```
char ch;
char next_char();

if ((ch = next_char()) != '\n') /* ... */;
```

Use of this notation is a matter of personal style. Be careful, however, that compactness does not slip over the edge into obscurity.

The compound assignment operator also provides a measure of notational compactness. For example,

```
int arraySum( int ia[], int sz ) {
    int sum = 0;
    for ( int i = 0; i < sz; ++i )
        sum += ia[ i ];
    return sum;
}
```

The general syntactic form of the compound assignment operator is

```
a op= b;
```

where `op=` may be one of the following ten operators: `+=`, `-=`, `*=`, `/=`, `%=`, `<<=`, `>>=`, `&=`, `^=`, `|=`. Each compound operator is equivalent to the following "longhand" assignment:

```
a = a op b;
```

The longhand notation for summing a vector, for example, is

```
sum = sum + ia[ i ];
```

Exercise 2-1. The following is illegal. Why? How would you correct it?

```
main() {
    int i = j = k = 0;
}
```

□

Exercise 2-2. The following is also illegal. Why? How would you correct it?

```
main() {
    int i, j;
    int *ip;

    i = j = ip = 0;
}
```

□

BEST AVAILABLE COPY

2.5 Incre

The increm
notational
operators s
be an lvalu

mai

}

Let's ill
defining a

1. push (v

2. pop (),

In addi

1. overflow

2. underfl

The int
The follow

1. size: t

2. array

3. top: a
top ec
size-

The pr

c

i

i

The ir
ment ope
must be p